Multi Sleeping Barber Problem

	Name	ID	Level	Department
1	اسلام ايمن زاكى عزب	202000124	Three	Computer science
2	رانيا رفعت حامد عليوه	202000308	Three	Computer science
3	سميحة محمود حمزه الشاذلى	202000404	Three	Computer science
4	علياء زغلول السيد	202000577	Three	Information system
5	علية حازم حسن محمود	202000579	Three	Computer science
6	محمود عيد عويس احمد	202000852	Three	Information system
7	عبد الرحمن احمد عبدالفتاح احمد	201900401	Four	Information system

Introduction

- The ‘Sleeping Barber Problem’ is a classic inter-process communication and synchronization problem between multiple operating system processes.

- The problem is analogous to that of keeping a barber working when there are customers, resting when there are none, and doing so in orderly manner.

- The Sleeping barber problem is often attributed to Edsger Dijkstra, One of the most important pioneers in Computer Science.

Solution Pseudocode

```
CHAIRS<-- 5              // # of chairs for waiting customers
customers<-- 0           // # of customers waiting for service
barbers<-- 0             // # of barbers waiting for customers
mutex<-- 1               // for mutual exclusion
num_waiting<--0          // customer are waiting (not being cut)
While True do
    Sem_wait(customers)      // go to sleep if # of customers is 0
    Sem_wait(mutex)          // acquire access to "waiting"
    Waiting<--waiting-1      // decrement count of waiting customers
```

```
   Sem_post(barbers)        // one barber is now ready to cut hair
   Sem_post(mutex)          // release 'waiting'
   Cut_hair()               // cut hair (outside critical region)
End
Sem_wait(mutex)             // enter critical region
If Waiting<CHAIRS Then      // if there are no free chairs, leave
   Waiting<--waiting+1      // increment count of waiting customers
   Sem_post(customers)      // wake up barber if necessary
    Sem_post(mutex)         // release access to 'waiting'
Sem_wait(barbers)           // go to sleep if # of free barbers is 0
get_haircut()               // be seated and be served
End
Sem_post(mutex)             // shop is full; do not wait
```

Explain of project :

1. The ‘Sleeping Barber Problem’ consist of one barber, one barber’s chair in a cuting room and a waiting room containing a number of chairs in it .

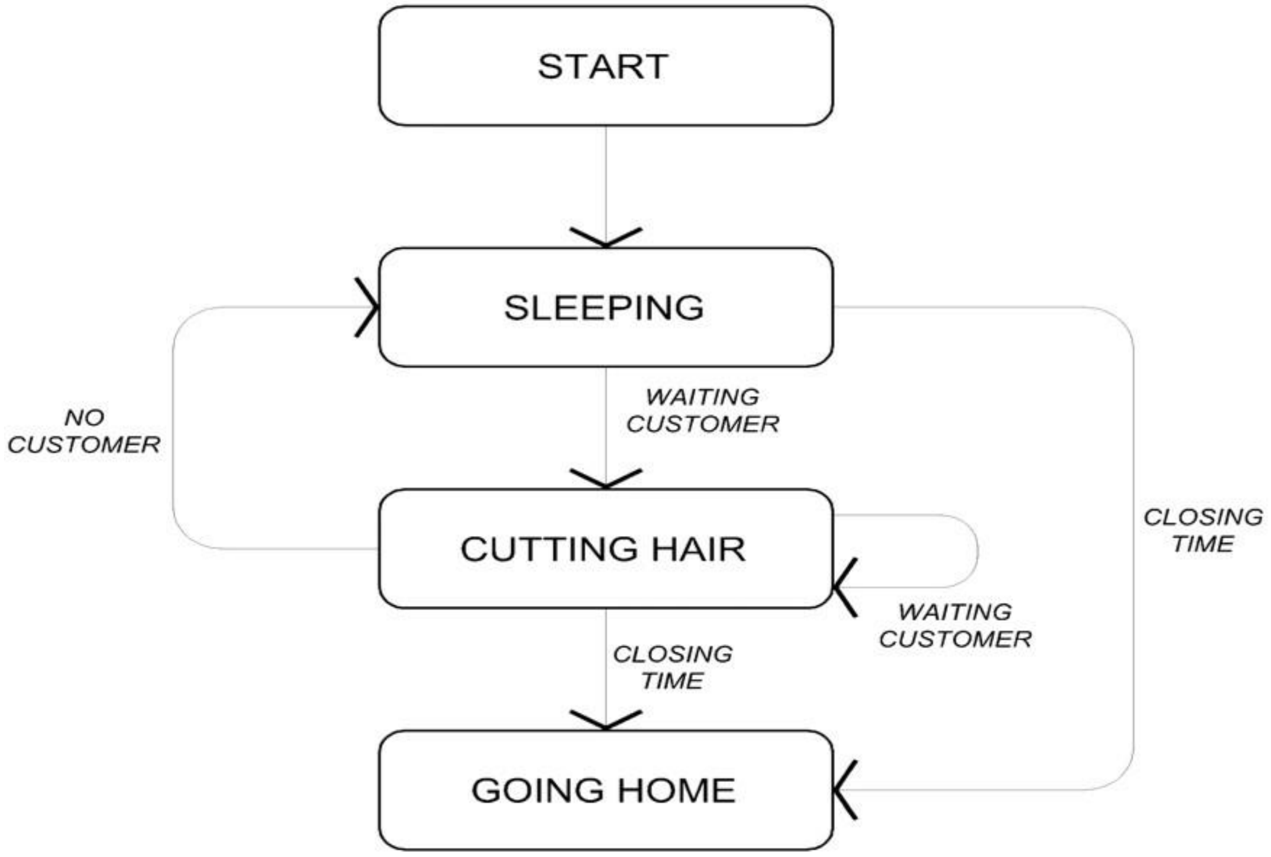
2. Each customer, when they arrive, looks to see what the barber is doing, If the barber is sleeping the customer wakes him up and sits in the cutting room chair .

3. If the barber is busy cutting hair, the customer stays in the waiting room and waits for their turn .

4. When the barber finishes cutting a customer’s hair, he dismisses the customer and goes to the waiting room to see if there are others waiting, if there are, he brings one of them back to the chair and cuts their hair, if there are none, he returns to the chair and sleeps in it .

Flow chart

- Here is a simple flowchart for the sleeping barber problem



Deadlock

Problem:

First:

There might be a scenario in which the customer ends up waiting on a barber and a barber waiting on the customer, which would result to a deadlock.

Second:

another problem may occur when two customers arrive at the same time when there is only one empty seat in the waiting room and both try to sit in the single chair.

Solution:

The solution to this problem includes three semaphores:

First:

is for the customer which counts the number of customers present in the waiting room (customer in the barber chair is not included because he is not waiting).

Second:

the barber 0 or 1 is used to tell whether the barber is idle or is working.

Third:

mutex is used to provide the mutual exclusion which is required for the process to execute which allows the customer to get exclusive access to the number of free seats and allows them to increase or decrease the number.

Starvation

Problem:

- Starvation might happen to a customer who is waiting for a long time when the customers don’t follow order or when the barber calls out a customer randomly.

- Since customer 2 was not serviced for a long period of time, starvation may occur and the ignored customer might decide to leave.

Solution:

The problem of starvation can be solved by making a queue where customers are added as they arrive, so that the barber may serve them on a first come first served as in first in first out.

Real World Application

The Clinic

Definition:

The problem is analogous to that keeping a doctor working when there are patients, resting and reading a book when there are none, and doing so in orderly manner.

Conditions:

1. Each patient, when they arrive, looks to see what the doctor is doing, if the doctor is reading, the patient call him up and sits so he can examine him.

2. If the doctor is busy seeing the patient, the patients stay in the waiting room and waits for their turn.

3. When the doctor finishes examining the patient, he dismisses the patient and goes to the waiting room to see if there are others waiting, if there are, he brings one of them back to the examining room, if there are none, he returns to his chair and read.

4. If a patient enters the clinic and all chairs are occupied, the patient leaves the clinic.

Deadlock Example and Solution:

A patient walks in and sees the doctor still examining the other patient, while waiting, he went for a comfort room. Simultaneously, the doctor finishes examining the first patient and checks if a patient is waiting at the lobby so it might end up the patient is waiting for a doctor and a doctor waiting on the patient which would result to a deadlock.

The solution to this problem includes three semaphores:

First:

is for the patient which counts the number of patients present in the waiting room.

Second:

the doctor 0 or 1 is used to tell whether the doctor is idle or is working.

Third:

mutex is used to provide the mutual exclusion which is required for the process to execute which allows the patients to get exclusive access to the number of free seats and allows them to increase or decrease the number.

Starvation Example and Solution:

it might happen when the patients don’t follow an order, as some people won’t get to see the doctor even though they had been waiting long. To solve this problem, we are utilizing a queue where patients are added as they arrive.